IEEE 754: What it means for humanity and your computer

The textbook[1] has an idealized view of floating point, which is wise. However, in this separate document I lay out some basic details of how floating point numbers are *actually* implemented on a computer, assuming they conform to the IEEE 754 standard, which they almost certainly do.

- A *bit* is a binary digit, the irreducible atom of memory, always in either of two states $\{0, 1\}$. Computer memories are organized into *bytes*, that is, groups of 8 bits.

- *Integers* are represented on computers using 1, 2, 4, or 8 bytes. It is straightforward, and exact if the integer is not too big in magnitude, but we ignore the details.

- The IEEE[2] 754 standard regards how *real* numbers are approximately represented in memory, that is, how *floating point* numbers are represented. "Floating point" is essentially just scientific notation, but using only finitely-many bits, so only a finite subset of real numbers can be represented. For more information on the standard than described here see
  `en.wikipedia.org/wiki/IEEE_754`

- The most important floating point representations use 32 or 64 bits, i.e. 4 pr 8 bytes. These are called binary32 (also known as `single`) and binary64 (`double`) in the standard. For binary32 the number
  $$x = (-1)^s \times (1.d_1d_2d_3\ldots d_{23})_2 \times 2^{(e_1\ldots e_8)_2 - 127}$$
  is represented by 32 bits this way:

s	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}	d_{17}	d_{18}	d_{19}	d_{20}	d_{21}	d_{22}	d_{23}

  In binary64 (`double`) the number
  $$x = (-1)^s \times (1.d_1d_2d_3\ldots d_{52})_2 \times 2^{(e_1\ldots e_{11})_2 - 1023}$$
  is represented by 64 bits this way:

s	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	$\cdots$	d_{51}	d_{52}

- Note that the "1." in the above representations, which appears before the d_i bits, is always present and therefore it does *not* use a bit of memory! It is called the "implicit leading bit".

- Unlike the system $\mathbb{F}$ in the textbook (Lecture 13), in any actual floating-point representation, including IEEE 754, there are only finitely-many allowed values of the exponent e. Thus there are only finitely-many representable floating point numbers.

- The IEEE 754 standard uses abstract language to describe the way the digits are arranged. Every representable *nonzero* number is of the form

(1) $$x = (-1)^s \times \frac{m}{\beta^{t-1}} \times \beta^e$$

[1] L. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM Press, 1997.

[2] IEEE = Institute of Electrical and Electronics Engineers.

for fixed positive integers β (the *base*) and t (the *precision*). The other symbols are $s \in \{0, 1\}$ (the *sign*), an integer m (the *mantissa*), and an integer e (the *exponent*). These satisfy

$$\beta^{t-1} \le m \le \beta^t - 1, \qquad e_{min} \le e \le e_{max}. \tag{2}$$

- In the current version of the standard, IEEE 754-2019, there are a number of formats. However, we ignore the *decimal* standards with $\beta = 10$, which are rarely used. We consider only *binary* formats with $\beta = 2$. The formats that matter most use 16, 32, 64, or 128 bits. We have already shown how the first two are implemented in memory. In terms of (1) and (2) they follow this table:

name	common name	precision t	exponent bits	exponent bias	e_{min}	e_{max}
binary16	`half`	11	5	$2^4 - 1 = 15$	-14	+15
binary32	`single`	24	8	$2^7 - 1 = 127$	-126	+127
binary64	`double`	53	11	$2^{10} - 1 = 1023$	-1022	+1023
binary128	`quadruple`	113	15	$2^{14} - 1 = 16383$	-16382	+16383

- If you convert the precision and exponent limits to decimal you get these values:

name	decimal precision	decimal e_{max}	decimal e_{min}
binary16	3.31	4.51	-4.21
binary32	7.22	38.23	-37.93
binary64	15.95	307.95	-307.65
binary128	34.02	4931.77	-4931.47

- For *normal* numbers, in `single` the standard requires $(e_1 \dots e_8)_2 \in \{1, 2, \dots, 254\}$ and in `double` the standard requires $(e_1 \dots e_{11})_2 \in \{1, 2, \dots, 2046\}$.

- If all bits e_i are zero or all bits e_i are one then the number has special/exceptional meaning. Note that the number zero is *not* in form (1), and it is such an exception. Zero is represented by setting all bits other than s to zero. Because the sign bit is not determined, this means "$+0$" and "-0" exist as separate representations. (Strange but true!)

- For a "subnormal" number, all the bits e_i are zero but some bits d_i are nonzero.

- When all bits e_i are one there are representations of $+\infty$ and $-\infty$ and of things that are "not a number" (`NaN`). We omit all details here.

- It is safe to assume your laptop implements IEEE-compliant binary64 floating point operations *in hardware*. Other types are commonly implemented in software, especially binary128, which is thus much slower on current computers. The smaller binary16 and binary32 formats are typically used for *storing* numbers, which increases storage capacity, or for graphics, or for the weights in a neural network.

- One major goal of the IEEE 754 standard is that axiom (13.5) in the textbook applies. The standard also addresses the rounding errors which occur in arithmetic operations (addition, multiplication, etc.), with the goal that axiom (13.7) in the textbook applies. In fact, the design goal is that the two axioms hold with as small a value for $\epsilon_{\text{machine}}$ as practically possible. Taking this viewpoint, the other details are not so important.